Complexidade de Algoritmos

Profª. Barbara Quintela
Prof. Jose J. Camata
Prof. Marcelo Caniato

barbara@ice.ufjf.br
camata@ice.ufjf.br
marcelo.caniato@ice.ufjf.br

ufjf

Algoritmo como Tecnologia

Pense na seguinte frase:

Suponha que os computadores fossem infinitamente rápidos e que a memória do computador fosse gratuita. Você teria alguma razão para estudar algoritmos?

Algoritmo como Tecnologia

Pense na seguinte frase:

Suponha que os computadores fossem infinitamente rápidos e que a memória do computador fosse gratuita. Você teria alguma razão para estudar algoritmos?

A resposta deve ser SIM!!!!

Verificar que seu método termina, e o faz com a resposta correta

Algoritmo como Tecnologia

- Computadores podem ser rápidos mas não são infinitamente rápidos!!
- Limitantes:
  - Hardware
  - Custo ($$$)
  - Gasto Energético
- **Tempo de processamento** é um recurso **limitado**, bem como **espaço** de **memória**.
  - **Necessário aliar estruturas de dados adequadas com algoritmos eficientes**

Analisando um algoritmo qualquer….

- Analisar um algoritmo significa **prever** os recursos de que o algoritmo necessita.
- **Qual é o custo de usar um dado algoritmo para resolver um problema específico?**
  - Características que devem ser investigadas:
    - análise do número de vezes que cada parte do algoritmo deve ser executada,
    - estudo da quantidade de memória necessária.

Analisando uma classe de algoritmo

- Qual é o algoritmo de menor custo possível para resolver um problema particular?
  - Toda uma família de algoritmos é investigada.
  - Procura-se identificar um que seja o melhor possível.
  - Coloca-se limites para a complexidade computacional dos algoritmos pertencentes à classe.

Funções de Complexidade

- Para medir o custo de execução de um algoritmo é comum definir uma função de custo ou função de complexidade *g*.
- Pode ser:
  - Função de **complexidade de tempo**: *g(n)* mede o tempo necessário para executar um algoritmo em um problema de tamanho *n*.
  - Função de **complexidade de espaço:** *g(n)* mede a memória necessária para executar um algoritmo em um problema de tamanho *n*.
    - OBS.: Utilizaremos *g* para denotar uma função de complexidade de tempo daqui para a frente.

Exemplo - Maior Elemento

Considere o algoritmo para encontrar o maior elemento de um vetor de inteiros *v[0,1,..., n-1], n ≥ 1.*

```
int max(int V[], int n)
{
      int max = v[0];
      for(int i=1;i < n; i++)
            if(v[i]>max)
                  max = v[i];
    return max;
}
```

Seja ***g* uma função de complexidade** tal que *g(n)* é o ***número de comparações entre os elementos*** de *v*, se *v* contiver *n* elementos.

Exemplo - Maior Elemento

Considere o algoritmo para encontrar o maior elemento de um vetor de inteiros *v[0,1,..., n-1], n ≥ 1.*

```
int max(int V[], int n)
{
      int max = v[0];
      for(int i=1;i < n; i++)
            if(v[i]>max)
                  max = v[i];
    return max;
}
```

Teremos n-1 comparações

Exemplo - Maior Elemento

Considere o algoritmo para encontrar o maior elemento de um vetor de inteiros *v[0,1,..., n-1], n* ≥ *1.*

```
int max(int V[], int n)
{
      int max = v[0];
      for(int i=1;i < n; i++)
            if(v[i]>max)
                  max = v[i];
    return max;
}
```

Logo $g(n) = n-1$

Observações

- A medida do custo de execução de um algoritmo **depende** principalmente do **tamanho da entrada** dos dados.
- Mas para alguns algoritmos, o custo de execução é uma função da entrada particular dos dados, não apenas do tamanho da entrada.
- Exemplos:
  - Para um algoritmo de ordenação se os dados de entrada já estiverem quase ordenados, então o algoritmo pode ter um custo menor.

Melhor Caso, Pior Caso e Caso Médio

- **Melhor caso:** menor tempo de execução sobre todas as entradas de tamanho *n*.
- **Pior caso:** maior tempo de execução sobre todas as entradas de tamanho *n.*
- **Caso médio** (ou caso esperado): média dos tempos de execução de todas as entradas de tamanho *n.*

Exemplo:

```
// busca sequencial
for(int i=0; i<n; i++){
    if (v[i] == x) break;
}
```

Melhor caso: se a chave está na primeira posição, apenas uma comparação, v[0] == x

Pior caso: sendo todos os elementos igualmente prováveis, comparar todos os valores até o final, n

Caso médio: média de comparação, (1+n)/2

Comportamento Assintótico

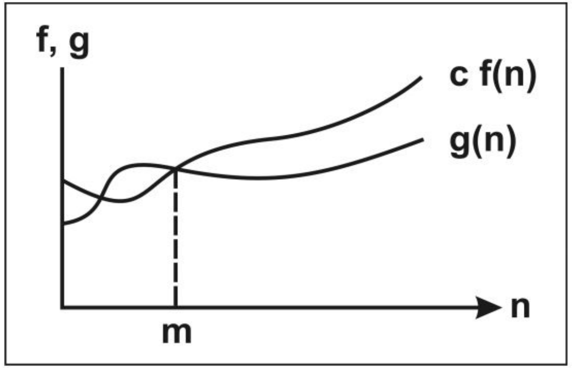
- O **parâmetro *n*** fornece uma **medida da dificuldade** para se resolver o problema.
- Para valores suficientemente pequenos de *n*, qualquer algoritmo custa pouco para ser executado, mesmo os ineficientes.
- A análise de algoritmos é realizada para valores grandes de *n (quando tende ao infinito)*.
- O **comportamento assintótico** de *g(n)* representa o **limite do comportamento do custo quando *n* cresce**.

Notações Assintóticas

- Fornecem um **vocabulário** para discutir o projeto e a análise de algoritmos
  - suficientemente **amplo** - ignora detalhes que dependem de arquitetura, escolha da linguagem de programação, compilador etc.
  - suficientemente **preciso** - permite comparações úteis entre abordagens algorítmicas de alto nível distintas para resolver problemas

Notação Big-O

- Escrevemos *g(n) = O(f(n))* para expressar que ***f(n)* domina assintoticamente *g(n)***. Lê-se *g(n)* é da ordem no máximo *f(n)*.
- Exemplo gráfico de dominação assintótica que ilustra a **notação Big-*O***



O valor da constante *m* é o menor possível, mas qualquer valor maior é válido

Definição: Uma função *g(n)* é *O(f(n))* se existem duas constantes positivas c e m tais que $g(n) \leq cf(n)$, para todo $n \geq m$

Exemplos Notação Big-O

- **Exemplo 1:** $g(n) = (n + 1)^2$
  - $g(n)$ é $O(n^2)$, quando $m = 1$ e $c = 4$
  - Isto porque $(n+ 1)^2 \leq 4n^2$ para $n \geq 1$.

Exemplo 1:

$(n+1)^2 \leq c\,n^2 \Rightarrow$

$(n^2+2n+ 1) \leq c\,n^2 \Rightarrow$

$(1 + 2/n + 1/n^2)n^2 \leq c\,n^2 \Rightarrow$

$(1 + 2/n + 1/n^2) \leq c$

Note que, para $n \geq 1$, $c=4$ satisfaz a desigualdade acima

- **Exemplo 2**: $g(n) = n$ e $h(n) = n^2$.
  - Sabemos que $g(n)$ é $O(n^2)$, pois para $n \geq 0$, $n \leq n^2$.
  - Entretanto $h(n)$ não é $O(n)$.
    - Suponha que existam constantes c e m tais que para todo $n \geq m$, $n^2 \leq cn$.
    - Logo $c \geq n$ para qualquer $n \geq m$, e não existe uma constante c que possa ser maior ou igual a n para todo n.

Classes de Comportamento Assintótico

- Em geral, é interessante **agrupar** os algoritmos / problemas **em Classes de Comportamento Assintótico**, que vão determinar a complexidade inerente do algoritmo
- Podemos avaliar programas comparando as funções de complexidade, negligenciando as constantes de proporcionalidade.
- Um programa com tempo $O(n)$ é melhor que outro com tempo $O(n^2)$.
  - Porém, as constantes de proporcionalidade podem alterar esta consideração.

Comparação de Programas

- **Exemplo:** um programa leva *100n* unidades de tempo para ser executado e outro leva $2n^2$. Qual dos dois programas é melhor?
  - Para $n < 50$, o programa com tempo $2n^2$ é melhor do que o que possui tempo *100n*.
  - Para problemas com entrada de dados pequena é preferível usar o programa cujo tempo de execução é $O(n^2)$.
  - Entretanto, quando *n* cresce, o programa com tempo de execução $O(n^2)$ leva muito mais tempo que o programa $O(n)$.

Principais Classes de Problemas

- ***O(1)***
  - Algoritmos de complexidade *O(1)* são ditos de **complexidade constante**.
  - Uso do algoritmo independe de *n*.

- ***O(log n)***
  - Um algoritmo de complexidade *O(log n)* é dito ter **complexidade logarítmica**
  - Típico em algoritmos que transformam um problema em outros menores.
  - Quando *n* é mil, $log_2 n \approx 10$, quando n é 1 milhão, $log_2 n \approx 20$

Principais Classes de Problemas

- ***O(n)***
  - Um algoritmo de complexidade *O(n)* é dito ter **complexidade linear.**
  - Em geral, um pequeno trabalho é realizado sobre cada elemento de entrada.
  - É a melhor situação possível para um algoritmo que tem de processar/produzir n elementos de entrada/saída.
  - Cada vez que n dobra de tamanho, o tempo de execução dobra.

- ***O(n log n)***
  - Típico em algoritmos que quebram um problema em outros menores, resolvem cada um deles independentemente e ajuntando as soluções depois.
  - Quando n é 1 milhão, $n\log_2 n$ é cerca de 20 milhões.
  - Quando n é 2 milhões, $n\log_2 n$ é cerca de 42 milhões, pouco mais do que o dobro.

Principais Classes de Problemas

- ***O(n^2)***
  - Um algoritmo de complexidade *O(n^2)* é dito ter **complexidade quadrática.**
  - Ocorrem quando os itens de dados são processados aos pares, muitas vezes em um laço dentro de outro.
  - Sempre que *n* dobra, o tempo de execução é multiplicado por 4.

- ***O(n^3)***
  - Um algoritmo de complexidade *O(n^3)* é dito ter **complexidade cúbica.**
  - Úteis apenas para resolver pequenos problemas.
  - Sempre que n dobra, o tempo de execução fica multiplicado por 8.

Principais Classes de Problemas

- ***$O(2^n)$***
  - Um algoritmo de complexidade $O(2^n)$ é dito ter **complexidade exponencial.**
  - Ocorrem na solução de problemas quando se usa força bruta para resolvê-los
  - Geralmente não são úteis sob o ponto de vista prático.
  - Quando *n* é 20, o tempo de execução é cerca de 1 milhão. Quando *n* dobra, o tempo fica elevado ao quadrado

- ***$O(n!)$***
  - Um algoritmo de complexidade $O(n!)$ também é dito ter **complexidade exponencial**.
  - Geralmente ocorrem quando se usa **força bruta** na solução do problema.

Comparação de Funções de Complexidade

Classe		Número de complexidade de operações e tempos de execução (1 instr/μseg)					
n		10		10^2		10^3	
constante	$O(1)$	1	1 μseg	1	1 μseg	1	1 μseg
logarítmico	$O(\lg n)$	3,32	3 μseg	6,64	7 μseg	9,97	10 μseg
linear	$O(n)$	10	10 μseg	10^2	100 μseg	10^3	1 mseg
$O(n \lg n)$	$O(n \lg n)$	33,2	33 μseg	664	664 μseg	9970	10 mseg
quadrático	$O(n^2)$	10^2	100 μseg	10^4	10 mseg	10^6	1 seg
cúbico	$O(n^3)$	10^3	1 mseg	10^6	1 seg	10^9	16,7 min
exponencial	$O(2^n)$	1024	10 mseg	10^{30}	$3{,}17 * 10^{17}$ anos	10^{301}	

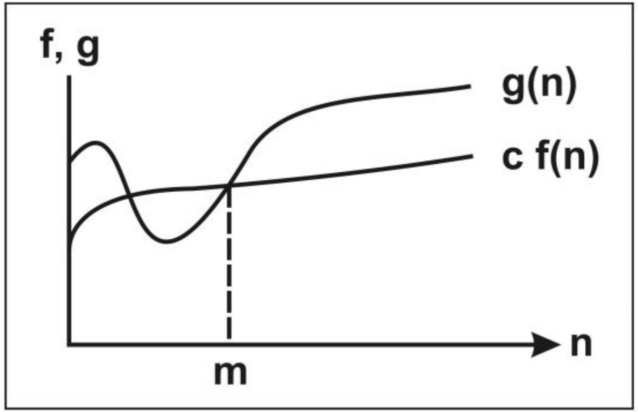
Comparação de Funções de Complexidade

n		10^4		10^5		10^6	
constante	$O(1)$	1	1 μseg	1	1 μseg	1	1 μseg
logarítmico	$O(\lg n)$	13,3	13 μseg	16,6	7 μseg	19,93	20 μseg
linear	$O(n)$	10^4	10 mseg	10^5	0,1 seg	10^6	1 seg
$O(n \lg n)$	$O(n \lg n)$	$133 * 10^3$	133 mseg	$166 * 10^4$	1,6 seg	$199{,}3 * 10^5$	20 seg
quadrático	$O(n^2)$	10^8	1,7 min	10^{10}	16,7 min	10^{12}	11,6 dias
cúbico	$O(n^3)$	10^{12}	11,6 dias	10^{15}	31,7 anos	10^{18}	31,709 anos
exponencia	$O(2^n)$	10^{3010}		10^{30103}		10^{301030}	

Outras Notações Importantes

Notação Big-Omega Ω

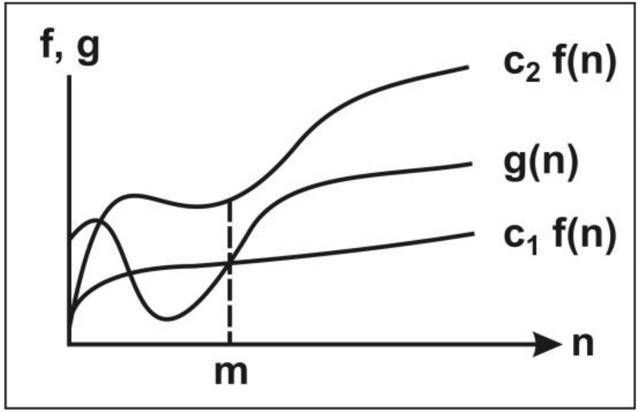
- ***g(n)*** é **Ω**(***f(n)***) se se existir uma constante real positiva *c* e uma constante inteira positiva *m* tais que ***g(n)*** ≥ *c* ***f(n)*** para n ≥ m.
- Exemplo:
  - $g(n) = 2n^2 + 3n + 1$ é $\Omega(n^2)$ para todo $n > 0$ e $c \leq 2$.
- Ao contrário do **Big-O**, toma-se a maior classe de funções possível
  - Diz-se que $2n^2 + 3n + 1$ é $\Omega(n^2)$, embora **f(n) = n log n**, **log n** e **1** sejam válidas.
  - Já $g(n) = n^3$, $f(n) = n^4$, etc não satisfazem a definição.



Outras Notações Importantes

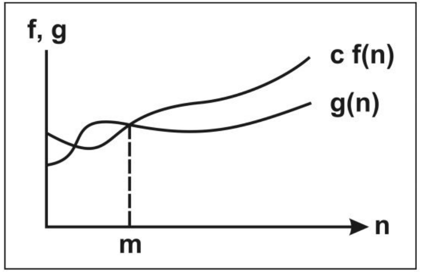
Notação Big-Theta Θ

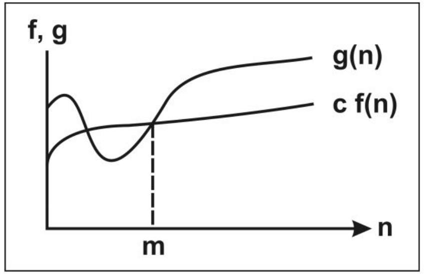
- ***g(n)*** é Θ(***f(n)***) se se existirem duas constantes reais positivas c_1 e c_2, e uma constante inteira positiva *m* tais que $c_1\, f(n) \le g(n) \le c_2\, f(n)$ para $n \ge m$.
- Em outras palavras: $g(n) = \Theta(f(n))$ se for ao mesmo tempo Ω(f(n)) e **O**(f(n))
- Exemplo:
  - $g(n) = 5n^2$ é $\Theta(n^2)$ para $c_1 = c_2 = 5$ e quaisquer valor de n;
- Observação:
  - qualquer polinômio de ordem d é $\Theta(n^d)$.

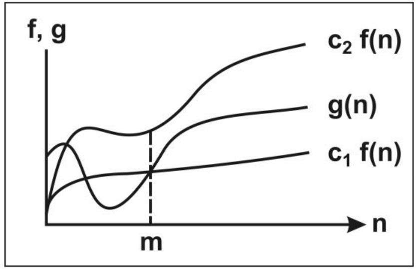


Em resumo ...

- **Big-O**
  - *g(n)* é **O**(*f(n)*) se *g(n)* cresce a **uma taxa menor ou igual** a *f(n)*.
- **Big-Omega**
  - *g(n)* é Ω(*f(n)*) se *g(n)* cresce a **uma taxa maior ou igual** a *f(n)*.
    - *Nota: Nota: g(n) é* Ω*(f(n)) se e somente se f(n) é* **O***(g(n))*
- **Big-Theta**
  - *g(n) é* Θ*(f(n))* se for **Big-O e Big-Omega ao mesmo tempo**
  - f(n) e g(n) crescem a uma mesma taxa assintótica.
    - *Nota: g(n) é* Θ*(f(n)) se e somente se f(n) é* Θ*(g(n))*







Exercícios

1. Agora que as notações já foram definidas, qual a complexidade temporal para um algoritmo de busca sequencial considerando um vetor de tamanho n?

a) O(1)
b) O(log n)
✓ c) O(n)
d) $O(n^2)$

Exercícios

2. E se fizermos buscas em dois vetores distintos?

a) O(1)
b) O(log n)
✓ c) O(n)
d) $O(n^2)$

```
bool busca(int vetA[], int vetB[], int n, int x)
{
    for(int i=0; i<n; i++)
        if(vetA[i] == x)
            return true;

    for(int i=0; i<n; i++)
        if(vetB[i] == x)
            return true;

    return false;
}
```

Exercícios

3. Qual a complexidade temporal do código a seguir?

a) O(1)
b) O(log n)
c) O(n)
✓ d) O(n^2)

```
bool confere(int vetA[], int vetB[], int n)
{
    for(int i=0; i<n; i++)
        for(int j=0; j<n; j++)
            if(vetA[i] == vetB[j])
                return true;

    return false;
}
```

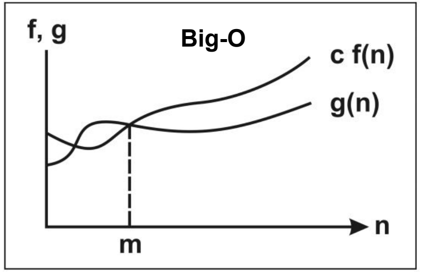
Exercícios

4. Qual a complexidade temporal do código a seguir?

a) $O(1)$
b) $O(\log n)$
c) $O(n)$
✓ d) $O(n^2)$

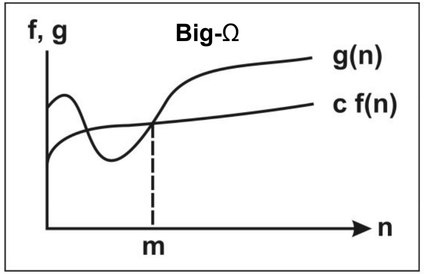
```
bool confere(int vetA[], int n)
{
    for(int i=0; i<n; i++)
        for(int j=i+1; j<n; j++)
            if(vetA[i] == vetA[j])
                return true;

    return false;
}
```

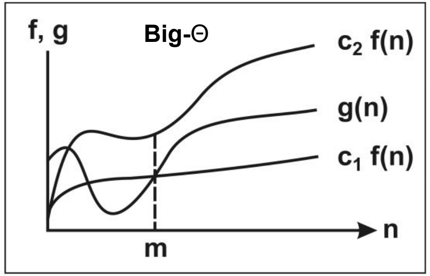
Exercícios



Utilizando as definições para as notações assintóticas, prove se são **verdadeiras** ou **falsas** as seguintes afirmativas:

1. $3n^3 + 2n^2 + n + 1 = O(n^3)$
2. $7n^2 \;= O(n)$
3. $2^n+2 = O(2^n)$
4. $2^{2n} \;= O(2^n)$
5. $5n^2 + 7n \;= \Theta(n^2)$
6. $6n^3 + 5n^2 \neq \Theta(n^2)$
7. $9n^3 + 3n \;= \Omega(n)$





Exercícios

Utilizando as definições para as notações assintóticas, prove se são verdadeiras ou falsas as seguintes afirmativas:

1. $3n^3 + 2n^2 + n + 1 = O(n^3)$ ✅
2. $7n^2 = O(n)$
3. $2^n+2 = O(2^n)$
4. $2^{2n} = O(2^n)$
5. $5n^2 + 7n = \Theta(n^2)$
6. $6n^3 + 5n^2 \neq \Theta(n^2)$
7. $9n^3 + 3n = \Omega(n)$

Pela definição do Big-O:
$3n^3 + 2n^2 + n + 1 <= c * n^3$ (dividindo por n^3)
$3 + 2/n + 1/n^2 + 1 <= c$
menor valor possível de c: $3 + 2 + 1 + 1 = 7$
Logo, é verdade para $c = 7$ e $m = 1$

Exercícios

Utilizando as definições para as notações assintóticas, prove se são verdadeiras ou falsas as seguintes afirmativas:

1. $3n^3 + 2n^2 + n + 1 = O(n^3)$ ✅
2. $7n^2 = O(n)$ ❌
3. $2^{n+2} = O(2^n)$
4. $2^{2n} = O(2^n)$
5. $5n^2 + 7n = \Theta(n^2)$
6. $6n^3 + 5n^2 \neq \Theta(n^2)$
7. $9n^3 + 3n = \Omega(n)$

Pela definição do Big-O:
$7n^2 <= c * n$
$7n <= c$
não existe um valor para c que seja sempre maior que n, portanto a afirmação é **falsa**

Exercícios

Utilizando as definições para as notações assintóticas, prove se são verdadeiras ou falsas as seguintes afirmativas:

1. $3n^3 + 2n^2 + n + 1 = O(n^3)$ ✅
2. $7n^2 = O(n)$ ❌
3. $2^{n+2} = O(2^n)$ ✅
4. $2^{2n} = O(2^n)$
5. $5n^2 + 7n = \Theta(n^2)$
6. $6n^3 + 5n^2 \neq \Theta(n^2)$
7. $9n^3 + 3n = \Omega(n)$

Pela definição do Big-O:

$2^{n+2} \le c * 2^n$

$2^n * 2^2 \le c * 2^n$

$2^2 \le c$

verdadeira para m = 1 e c = 4

Exercícios

Utilizando as definições para as notações assintóticas, prove se são verdadeiras ou falsas as seguintes afirmativas:

1. $3n^3 + 2n^2 + n + 1 = O(n^3)$ ✅
2. $7n^2 = O(n)$ ❌
3. $2^{n+2} = O(2^n)$ ✅
4. $2^{2n} = O(2^n)$ ❌
5. $5n^2 + 7n = \Theta(n^2)$
6. $6n^3 + 5n^2 \neq \Theta(n^2)$
7. $9n^3 + 3n = \Omega(n)$

Pela definição do Big-O:
$(2^n)^2 <= c * 2^n$ (dividindo por 2^n)
$2^n <= c$
Logo, não existe um valor para c que seja sempre maior que n, portanto a afirmação é **falsa**

Exercícios

Utilizando as definições para as notações assintóticas, prove se são verdadeiras ou falsas as seguintes afirmativas:

1. $3n^3 + 2n^2 + n + 1 = O(n^3)$ ✅
2. $7n^2 = O(n)$ ❌
3. $2^{n+2} = O(2^n)$ ✅
4. $2^{2n} = O(2^n)$ ❌
5. $5n^2 + 7n = \Theta(n^2)$ ✅
6. $6n^3 + 5n^2 \neq \Theta(n^2)$
7. $9n^3 + 3n = \Omega(n)$

Por definição de Big-Θ:

$5n^2 + 7n <= c2 * n^2$	$c1 * n^2 <= 5n^2 + 7n$
$5 + 7/n <= c2$	$c1 <= 5 + 7/n$
$5 + 7/1 <= c2$	$c1 <= 5 + 7/\text{infinito}$

verdadeiro para $m = 1$, $c1 = 5$ e $c2 = 12$
no cálculo de c1, considera-se m tendendo ao infinito fazendo 7/n assumir o menor valor possível; já no cálculo de c2, considera-se m = 1 para que 7/n assuma o maior valor possível.

Exercícios

Utilizando as definições para as notações assintóticas, prove se são verdadeiras ou falsas as seguintes afirmativas:

1. $3n^3 + 2n^2 + n + 1 = O(n^3)$ ✅
2. $7n^2 = O(n)$ ❌
3. $2^{n+2} = O(2^n)$ ✅
4. $2^{2n} = O(2^n)$ ❌
5. $5n^2 + 7n = \Theta(n^2)$ ✅
6. $\boxed{6n^3 + 5n^2 \neq \Theta(n^2)}$ ✅
7. $9n^3 + 3n = \Omega(n)$

Por definição de Big-Θ:

$c1 * n^2 <= 6n^3 + 5n^2$ $\quad$ $6n^3 + 5n^2 <= c2 * n^2$

$c1 <= 6n + 5$ $\quad$ $6n + 5 <= c2$

para m = 1, c1 pode valer 11. Porém, não existe um valor fixo para c2. Portanto, a afirmativa é **VERDADEIRA**, pois perguntava se a expressão era DIFERENTE de $\Theta(n^2)$.

Exercícios

Utilizando as definições para as notações assintóticas, prove se são verdadeiras ou falsas as seguintes afirmativas:

1. $3n^3 + 2n^2 + n + 1 = O(n^3)$ ✅
2. $7n^2 = O(n)$ ❌
3. $2^{n+2} = O(2^n)$ ✅
4. $2^{2n} = O(2^n)$ ❌
5. $5n^2 + 7n = \Theta(n^2)$ ✅
6. $6n^3 + 5n^2 \neq \Theta(n^2)$ ✅
7. $9n^3 + 3n = \Omega(n)$ ✅

Por definição de Big-Ω:
$9n^3 + 3n >= c * n$
$9n^2 + 3 >= c$
verdadeira para m = 1 e c = 12.

Referências

1. Slides baseados nos slides de Nivio Ziviani. Disponivel em http://www2.dcc.ufmg.br/livros/algoritmos/slides.php
2. LEISERSON, C. E.; STEIN, C.; RIVEST, R. L., CORMEN, T.H. Algoritmos: Teoria e Prática. Editora Campus, 2002. Segunda
    a. Cap. 1, Cap. 3
3. Cap 2 Livro Adam Drozdek - Estrutura de Dados e Algoritmos em C++ (disponível na biblioteca virtual)